NLP methods for crypto price prediction

Grigorii Kuzmin
gikuzmin@edu.hse.ru

Alexey Kipriyanov
aakipriyanov@edu.hse.ru

December 19, 2022

1 Idea and Motivation

Cryptocurrency markets are known to be strongly affected by public opinion and news when several tweets can make crypto price either rocket or plummet. Thus, the main objective of this project is to develop the tools to forecast the direction of cryptos' prices, using sentiment analysis of social networks such as Reddit. The obtained technique might be potentially applied in trading algorithms to estimate the noise traders' activity.

2 Data

'Reddit' is a mix of forum and social net: users may post texts, pictures, videos, etc., other users may comment and like/dislike the post.

Crypto data was collected from Yahoo finance, we took cruptocurrency such as Bitcoin, Ethereum, USDT, USDC, Binance Coin and Dogecoin.

The data covers the period for 1 calendar year (21-11-2021 to 22-11-2022). We took *title*, *body*, *id*, *score* (number of likes), *number of comments* and *creation time* of post in crypto-related groups. The analysis is built mainly on the titles, as body of post may be either url, or picture, or video, which is not parceable. Other variables have no missing values. However, we delete all $2 \times IQR$ outliers based on *'Score'* and *'Total comments'* features, leaving 39'066 observations in sample.

3 NLP methodology and results

There are mainly two approaches to analyze words: *Lexicon-based* methods and *ML supervised training* methods. In this project we focus on the latter category since lexicon methods though they are simple and intuitive demand field-specific dictionaries to make relevant sentiment conclusions. While ML supervised technique finds the sentiment scores not exante but via training set, although technically it is much more complicated and computationally intensive task

3.1 Words into numbers: TF-IDF and Word2Vec

TF-IDF

This method the sentiment of the word by combining its importance in the file of interest (**TF**) and uniqueness in all files considered (**IDF**).

$$tf(word, textfile) = \frac{\#\text{ of the word in the text file}}{\text{total }\#\text{ of all words in the text file}}$$

$$idf(word, othertextfiles) = log(\frac{\#\text{ of other text files}}{\#\text{ of text files with this word}})$$

Thus, we form the final $tf - idf$ statistic, evaluating the sentiment importance of the word:

$$tf - idf(word, textfile, othertextfiles) = tf \times idf$$

Word2Vec

This method tries to construct semantic similarity between the words and attaches each to semantic vector via **CBO** algorithm. Instead of evaluating numeric vectors for each word separately, we aggregate them into one vector for each separate post. Then these vectors are used to determine the degree of similarity by *cosine similarity* which is scalar product of two vectors multiplied by their distance:

$$d = \frac{xy}{||x|| \cdot ||y||}$$

Thus, d varies from -1 to 1 which implies difference and similarity in terms of usage, implying words might be antonyms, but are similar within the model as are used in the same context

4 Prediction Algorithms

Data Marking

Before moving to prediction, we mark our data. We will divide posts into three categories: negative, neutral and positive with respect to their effect on BTC returns. This division allows us to distinguish between the news that appear to lead to significant returns from the ones that are associated with almost zero returns (It is wrong to consider both 0.05% and 5% as positive returns as 0.05% is more comparable to -0.05%). Thus, neutral group captures posts that have positive or negative returns around zero

Table 1: Marking Thresholds

negative (-1)	**neutral (0)**	**positive (1)**
return $<$ -2%	return $\in$ [-2%; +2%]	return $>$ +2%

TDF-IDF

We apply **TF-IDF** algorithm on the data with posts' and then try to predict the directions of returns (markings) on the test data via SVM (Support Vector Machine). As measure of accuracy we use $F - score$. The expanding window scheme is applied to account for sentiment data accumulation

Word2Vec-KNN Prediction algorithm

We try to construct the sentiment prediction using **Word2Vec** words representation and **KNN** methodology. Our algorithm can be outlined as the following pseudocode:

For post $i = 1$ to $M + V$ (M - train sample, V - test sample):
Step 1: for $m = 1$ to M: if i is not from M, estimate cosine similarity (d) of post i and post m
Step 2: Select K highest d to obtain K nearest neighbours and take their initial sentiments (by initial sentiment we take the log return, associated with the post)
Step 3: via K nearest neighbours predict *sentiment* of a post t:

$$Sentiment_t = \frac{\sum_{\{m \in KNN\}} d_{im}^3 \times Initial\ Sentiment}{\sum_{\{m \in KNN\}} d_{im}^3}$$

Step 4: Estimate *sureness metric* (s_t) for our prediction t:

$$s_t = \frac{\sum_{\{m \in KNN\}} d_{im}^3}{K}$$

Step 5: Aggregating metrics for a day among all posts $t \in N$ (N - # of posts at the day):

$$s_{pred} = \frac{\sum_{t=1}^{N} s_t}{N} \qquad Sentiment_{pred} = \frac{\sum_{t=1}^{N} Sentimen_t \times s_t}{\sum_{t=1}^{N} s_t}$$

The logic of these metrics is as follows: in *Predicted sentiment* we estimate the weighted sentiment, by weighting via cosine similarity (d). We impose the power of 3 on d to penalize the low d values to distract from the case when we select K nearest neighbours, but all of them have very low relation to the new post (low d-s).

Table 2: Empitical Results

TF-IDF + SVM	**Word2Vec + KNN**
Train 80%, Test 20%	2 days to predict next day
Expanding window	Rolling window
F-beta=0.7027 MSE=1.3067	F-beta=0.6071 MSE=1.1429

Limitations and Improvements

- **TF-IDF**:
  - Add additional factors, such as posts length, binary for emoji-es and presence of video materials

  - Other alternative classification techniques could be used such as logit, Random Forests

- **Word2Vec**
  - Extremely computationally extensive, that is why, we should check the performance with the training data larger than posts within two days to predict sentiment of the third, however, current scheme tries to account that trends for cryptos become outdated very quickly, so we eliminate sentiments that have no longer effect on returns.

Potential applications:
Constructing trading algorithm, based on predicted marking:

- *positive* (1) $\rightarrow$ *Buy*
- *neutral* (0) $\rightarrow$ *Hold*
- *negative* (-1) $\rightarrow$ *Sell*

Performance could be compared to some other naive or more sophisticated strategies.

5 Competing methodologies and results

Polynomial regression

We implemented several methodologies based on purely financial data (though, pretty limited amount of data). The key metric of quality is out of sample MSE.

We use polynomial regression, random forest and GARCH (1,1) model.

Polynomial regression

As inputs lags of BTC, ETH, BNB and DOGE Returns,lags of BTC Volume and Range, Score and Total Comments are used. We perform test-validating-test scheme for the estimation (6-2-2). After rescaling of variables, cross-validating result shows that optimal power is 1 (linear regression). MSE is 0.9185.

Random Forest

Using the set same of variables grid search shows that optimal number of trees is 520. We also tried to tune different hyperparameters, but the result was mainly sensitive to the number of trees. Note since that we break down all time series structure with RF approach we use train-test split 8-2 to be consistent. MSE is much better: 0.0337.

GARCH(1,1)

We estimate GARCH(1,1) model since it is known to be superior in finance[1]. To forecast the OOS performance we randomly generate standard Normal shocks. MSE is 0.0464

[1]P. Reinhard Hansen and A. Lunde, "Does anything beat a GARCH(1,1)? A comparison based on test for superior predictive ability," 2003

ANN

Inspired by the novel methodology we decided to try it as well. Firstly, we estimated the ANN using forecasts of the model as inputs (preliminary we generated in-sample forecasts). The rationale for this is simple: if no model is extremely good can we combine their forecasts to increase the overall predictive ability? We estimate the ANN with 2 hidden layers, 4096 neurons each, one concatenating (hidden2 + input). MSE is 0.03507. Worse than RF alone.

ANN expanded

At the next stage we keep the structure of the ANN, but add more inputs: consider the full set of explanatory variable plus forecasts. This results in even higher MSE = 1.3806.

RNN

Finally, we tried recursive RNN to keep the track of the data. It alone with one RNN layer leads to MSE = 0.0248. Absolute winner[2].

6 Suggested improvements and further research

We consider that our work can be extended in several ways. First and the most foremost, we need to increase the data set (switch to Twitter parcing, as Reddit's abilities is fully exhausted). Secondly, a bit deeper ANN research and application is needed, simply – implement LSTM type of models that fits time series data better. As we mentioned earlier, we do not take into account the body of the post as it often contains pictures/GIFs/videos (memes mainly). It would be very interesting to try to analyse the sentiment of those elements.

[2]To prove this claim formally we need to perform Diebold-Mariano test with McCracken standard errors or Giacomini-White test. But let's leave it to S.Budanova classes